Lab 02

Avoid-Obstacle, Follow-Object and Random Wander Worksheet

Robot Name	_____________Walter________________________
Team Member Name	______________Yao Xiong_______________________
Team Member name	_________________Zhengyang Bi____________________

Purpose

In your own words, state the purpose of lab 02 in the following space.

In this lab, we will deploy a layered control algorithm for the robot. This control strategy combines multiple behaviors, starting with the most basic behavior (obstacle avoidance), and including random wandering, shy behavior, following behavior, and aggressive behavior.

Part I – Random Wander (Layer 1)

What was the general plan you used to implement the random wander and obstacle avoidance behaviors?

Obstacle avoidance is given the highest priority behavior. When sensors detect a high risk of collision, the vehicle will stop immediately. When there is no risk of collision, random wandering (at the higher layer) takes control of the robot.

Part II – Test the Sonar Sensors

Based upon your testing, what is the maximum and minimum range for the sonar sensors on your robot?

22 max, 0 min, in centimeters

What is one advantage of using a sonar sensor instead of an infrared sensor? In the following space, discuss the accuracy of the sensor.

One key advantage of sonar sensors over infrared sensors is their ability to work effectively in varying lighting conditions. Since sonar uses sound waves rather than light, it can operate reliably in complete darkness, bright sunlight, or changing light levels, whereas infrared sensors can be affected by ambient light interference or changes in lighting conditions.

Part III – Test the Infrared Lidar Sensors

Based upon your testing, what is the maximum and minimum range for the infrared lidar sensors on your robot?

30 max, 1 min in centimeter

What is one advance of using an infrared lidar instead of a sonar sensor? In the following space, discuss the accuracy of the sensor.

One key advantage of infrared lidar over sonar sensors is its superior spatial resolution and accuracy. Lidar can detect much finer details and create more precise 2D maps of an environment because light waves have a much shorter wavelength than sound waves. This allows lidar to distinguish between smaller objects and features that sonar might miss or blur together.

Part IV – Collide Behavior

What distance did you use for the collide behavior?

10 cm

Did you use the lidar, sonar or a combination of the two sensors? How did you decide?

All Lidars are used. Lidars are pointing at all directions on X Y axis of this vehicle. Therefore, it can avoid most collisions in the surrounding directions of this vehicle.

How could you implement collide behavior for sensors other than the front sensors?

If this is not a shy child mode, the sensors result will be dismiss unless launching obstacle avoidance.

Part V – Runaway Behavior (Layer 0)

How did you represent the feel force function on your robot?

The feel force's magnitude and direction are determined by the lidar measurements - its direction aligns with the lidar's measured direction to the obstacle, while its magnitude is proportional to the measured distance between the robot and the obstacle. In other words, both the size and orientation of the feel force are derived from the lidar's distance and directional readings of nearby obstacles.

How did the robot respond with respect to the force felt, note that it could move in reverse, or it could turn a certain angle, think about what makes most sense for potential field navigation.

The robot moves in a direction away from obstacles based on the feel forces. We calculate the net force by combining all individual feel forces into a resultant vector with both direction and magnitude. Since these feel forces are mapped to our robot's X and Y coordinate system, we can make the robot rotate to align with the net force direction and then move along this resulting force vector.

How does your robot handle getting the robot unstuck when the vectors sum to zero?

We identified several special conditions through our analysis. When the robot detects objects directly in front and behind it, or on both its left and right sides, or when it's surrounded by objects in all directions, the net force in either the X or Y axis may become zero. We treat these as special cases and have designed specific solutions for each scenario.

Part VI – Follow Behavior (Alternate Layer 0)

In your own words, describe how to implement proportional control to follow an object,

We calculate the error by subtracting the lidar sensor's measured distance from the threshold distance. To determine the running distance, we multiply this error by 0.5. If the error is negative, it means the robot is too close to the obstacle and should move backward by the calculated distance. If the error is positive, it means the robot is too far from the obstacle and should move forward by the calculated distance.

Are there situations where the robot would get stuck when following?

If so, how would the robot correct for that?

The robot can get stuck when its error approaches zero, causing it to oscillate back and forth around the threshold value. To address this, we established both an obstacle threshold and an obstacle margin for the robot. In the following behavior, if the robot's distance to our hand falls within this defined range, the robot will stop moving. Otherwise, the robot will use a proportional control (KP) method to determine its movement.

Part VII – Subsumption Architecture – Smart Wander Behavior (Layers 0 and 1)

What is the difference between on-off and proportional control?

In on-off control, the robot would either move at full speed or stop completely when approaching an obstacle. However, the proportional controller is a feedback controller, which enables that to adjust based on error detected by sensor. Proportional control makes robots move more smoothly.

How did feedback control improve the random wander and avoid obstacle state machine?

Feedback control allows the robot to receive the data continuously and process it. The movement can be more smoothly.

Part VIII – Subsumption Architecture – Smart Follow Behavior (Layers 0 and 1)

You robot currently runs either a smart wander or smart follow behavior. How could you create a state machine that integrates smart follow and smart wander? What type of input would trigger the avoid versus the follow behavior?

When there is no obstacle (front sensor senses nothing), the state will be smart wander. When a potential target is detected within the specified following range, it should switch to smart follow behavior. Obstacle avoidance remains the highest priority behavior that can override either state.

Conclusions

1. How well did your software design plans match the reality of what you implemented on the robot?

   The reality matches well with our design. However, the lidar sensors are unstable sometimes.

2. How well did your software design plans match the reality of how the robot performed? Compare it to the theory you learned in class.

   The theory matches well with the P controller. If we increase Kp, the oscillation in the following behavior becomes larger. If we reduce Kp into small value, the movement distance is too small.

3. How did you create a modular program and integrate the two layers into the overall program?

   Using different while loop (each state will have one) and different Boolean function for change in state. When the Boolean function for avoid-obstacle gives false, the while loop for avoid-obstacle shall not be called, random wandering is called. When the Boolean function gives true, random wandering will stop, state will jump to the obstacle avoidance. (each state will have its conditions for enter and jump out)

4. Did you use the sonar and IR sensors to create redundant sensing on the robot's front half?

   No, we only use Lidar sensors because the sonar is also unstable sometimes.

5. How could you create a smart wander routine to entirely cover a room? Similar to what a Roomba does.

   We use the random number to generate random distance and angle, and the robot will move to those angle and distance. The robot will stop and run away if it detects the obstacle. Once the robot finishes scanning the whole room, it will stop.

6. What kind of errors did you encounter with obstacle avoidance behavior?

   The lidar sensors were unstable, and we could not read the lidar data while combining with the motor movement.

7. How could you improve obstacle avoidance behavior?

   We added more delays in our program to prevent the program from being hugged, and we forced the lidar to send a pulse and receive it first before using the lidar to receive the data.

8. Were there any obstacles that the robot could not detect?

   Yes, the obstacles which are lower than the height of the sensor.

9. Were there any situations when the range sensors did not give you reliable data?

   Yes. Sometimes, because our robot speed was too high, the lidar sensors' data did not update very quickly.

10. How did you keep track of the robot's states in the program?

    Use print command to show the state.

11. Did the robot encounter any "stuck" situations? How did you account for those?

    Yes, the P controller for will cause this when doing following behavior.

12. What should the subsumption architecture look like for the addition of the go-to-goal and avoid- obstacle behaviors?

    The avoid-obstacle behaviors shall be set as a priority. The go-to-goal shall be in the second layer which is used to deal with the requirements for moving robots.

13. What did you learn?

    I have learned how to deploy a layered reactive control strategy on a robot.

14. What did you observe?

    The sensors’ data were unstable. It took us a lot of time to figure out how to use duel cores to read the lidar sensor correctly while moving the robot.

15. What questions do you still have?

    N/A

Appendix

Insert your properly commented, modular code in the following space.

```
/*
  Walter-Lab02.ino
  Yao Xiong & Zhengyang Bi 2024/1/17

  This program implements various autonomous behaviors for a wheeled robot using stepper motors and distance sensors.
  The robot features reactive behaviors including obstacle avoidance, runaway response, and following behavior.
  It uses a dual-core architecture (M7 and M4) for parallel processing of sensor data and motor control.

  The primary functions created are:
  goToAngle - turns robot to specified absolute angle in degrees
  goToGoal - moves robot to specified x,y coordinate in inches
  forward, reverse - both wheels move with same velocity, same direction
  spin - both wheels move with same velocity opposite direction
  smartWanderBehavior - random movement with obstacle avoidance
  runawayBehavior - moves away from detected obstacles
  followBehavior - maintains constant distance from detected obstacle
  collideBehavior - emergency stop when obstacle detected

  Sensor Integration:
  - 4 LIDAR sensors (front, back, left, right) for distance measurement
```

- 2 wheel encoders for position tracking
- PID control system for accurate movement

Core Architecture:

M4 Core - Handles sensor data collection and processing

M7 Core - Manages movement control and behavioral algorithms

Uses RPC (Remote Procedure Call) for inter-core communication

Hardware Connections:

Arduino pin mappings: https://docs.arduino.cc/tutorials/giga-r1-wifi/cheat-sheet#pins

A4988 Stepper Motor Driver Pinout: https://www.pololu.com/product/1182

Stepper Motor Control:

digital pin 48 - enable PIN on A4988 Stepper Motor Driver StepSTICK

digital pin 50 - right stepper motor step pin

digital pin 51 - right stepper motor direction pin

digital pin 52 - left stepper motor step pin

digital pin 53 - left stepper motor direction pin

Status LEDs:

digital pin 13 - enable LED on microcontroller

digital pin 5 - red LED in series with 220 ohm resistor

digital pin 6 - green LED in series with 220 ohm resistor

digital pin 7 - yellow LED in series with 220 ohm resistor

Sensors:

digital pin 18 - left encoder pin

digital pin 19 - right encoder pin

digital pin 8 - front LIDAR

digital pin 9 - back LIDAR

digital pin 10 - left LIDAR

digital pin 11 - right LIDAR

```
  Constants:
  Robot Physical Parameters:
  - Wheel radius: 1.7 inches
  - Robot width: 9.0 inches
  - Steps per revolution: 800

  Movement Parameters:
  - Default step speed: 300 steps/sec
  - Maximum speed: 1500 steps/sec
  - Maximum acceleration: 10000 steps/sec^2

  Sensor Parameters:
  - Maximum LIDAR distance: 40 cm
  - Obstacle threshold: 10 cm

  Required Libraries:
  - AccelStepper: https://www.airspayce.com/mikem/arduino/AccelStepper/
  Install via:
  Sketch->Include Library->Manage Libraries...->AccelStepper->Include
  OR
  Sketch->Include Library->Add .ZIP Library...->AccelStepper-1.53.zip

  See PlatformIO documentation for proper way to install libraries in Visual Studio
*/

//includew all necessary libraries
#include <Arduino.h>      //include for PlatformIO Ide
#include <AccelStepper.h>  //include the stepper motor library
#include <MultiStepper.h>  //include multiple stepper motor library
#include <math.h>
#include "RPC.h"
```

```
//state LEDs connections
#define redLED 5          //red LED for displaying states
#define grnLED 6          //green LED for displaying states
#define ylwLED 7          //yellow LED for displaying states
#define enableLED 13       //stepper enabled LED
int leds[3] = { 5, 6, 7 };  //array of LED pin numbers

//define motor pin numbers
#define stepperEnable 48  //stepper enable pin on stepStick
#define rtStepPin 50      //right stepper motor step pin
#define rtDirPin 51      // right stepper motor direction pin
#define ltStepPin 52      //left stepper motor step pin
#define ltDirPin 53      //left stepper motor direction pin

AccelStepper stepperRight(AccelStepper::DRIVER, rtStepPin, rtDirPin);  //create instance of right stepper
motor object (2 driver pins, low to high transition step pin 52, direction input pin 53 (high means
forward)
AccelStepper stepperLeft(AccelStepper::DRIVER, ltStepPin, ltDirPin);   //create instance of left stepper
motor object (2 driver pins, step pin 50, direction input pin 51)
MultiStepper steppers;                                    //create instance to control multiple steppers at the
same time

#define stepperEnTrue false  //variable for enabling stepper motor
#define stepperEnFalse true  //variable for disabling stepper motor
#define max_speed 1500      //maximum stepper motor speed
#define max_accel 10000      //maximum motor acceleration

int pauseTime = 2500;  //time before robot moves
int stepTime = 500;    //delay time between high and low on step pin
int wait_time = 1000;  //delay for printing data
```

```
//define encoder pins

#define LEFT 0                   //left encoder

#define RIGHT 1                  //right encoder

const int ltEncoder = 18;           //left encoder pin (Mega Interrupt pins 2,3 18,19,20,21)

const int rtEncoder = 19;           //right encoder pin (Mega Interrupt pins 2,3 18,19,20,21)

volatile long encoder[2] = { 0, 0 };  //interrupt variable to hold number of encoder counts (left, right)

int lastSpeed[2] = { 0, 0 };        //variable to hold encoder speed (left, right)

int accumTicks[2] = { 0, 0 };       //variable to hold accumulated ticks since last reset


#define TO_LEFT -1  // direction variables to left

#define TO_RIGHT 1  // direction variables to right


#define CLOCKWISE 1        //direction variables for clockwise motion

#define COUNTERCLOCKWISE -1  //direction variables for counterclockwise motion


// Helper Functions

const float pi = 3.14159;        //pi

const float wheelRadius = 1.7;     //robot wheel radius in inches

const int stepsPerRevol = 800;     //robot wheel steps per revolution

const float robotWidth = 9.0;      //robot width in inches

const int defaultStepSpeed = 300;  //robot default speed in steps per second


const int PIDThreshold = 50;  //PID threshold

const int PIDkp = 1;         //PID proportional gain

const int encoderRatio = 20;  //ratio between the encoder ticks and steps


// random move maximum values

const int maxTurnAngle = 90;      // Maximum turn angle in degrees

const int maxDistanceMove = 12.0;  // Maximum move distance in inches


// lidar constant values

#define FRONT 0
```

```
#define BANK 1

#define LEFT 2

#define RIGHT 3

#define numOfSens 4


uint16_t wait = 100;

int16_t ft_lidar = 8;

int16_t bk_lidar = 9;

int16_t lt_lidar = 10;

int16_t rt_lidar = 11;


int16_t lidar_pins[4] = { 8, 9, 10, 11 };


int16_t leftSnr = 3;

int16_t rightSnr = 4;


const int MAX_LIDAR_DISTANCE = 40;


// Check for obstacles - adjust these thresholds based on your needs

const int OBSTACLE_THRESHOLD = 10;  // centimeters


// a struct to hold lidar data

struct lidar {

  // this can easily be extended to contain sonar data as well

  int front;

  int back;

  int left;

  int right;

  // this defines some helper functions that allow RPC to send our struct (I found this on a random
forum)
```

```
  MSGPACK_DEFINE_ARRAY(front, back, left,
right);  //https://stackoverflow.com/questions/37322145/msgpack-to-pack-structures
https://www.appsloveworld.com/cplus/100/391/msgpack-to-pack-structures
} lidarData;

struct lidar read_lidars() {
  return lidarData;
}

// a struct to hold sonar data
struct sonar {
  // this can easily be extended to contain sonar data as well
  int left;
  int right;
  // this defines some helper functions that allow RPC to send our struct (I found this on a random
forum)
  MSGPACK_DEFINE_ARRAY(left, right);  //https://stackoverflow.com/questions/37322145/msgpack-to-
pack-structures https://www.appsloveworld.com/cplus/100/391/msgpack-to-pack-structures
} sonarData;

struct sonar read_sonars() {
  return sonarData;
}

struct lidar lidar_data;
struct sonar sonar_data;

//interrupt function to count left encoder tickes
void LwheelSpeed() {
  encoder[LEFT]++;  //count the left wheel encoder interrupts
}
```

```
//interrupt function to count right encoder ticks
void RwheelSpeed() {
  encoder[RIGHT]++;  //count the right wheel encoder interrupts
}

void allOFF() {
  for (int i = 0; i < 3; i++) {
    digitalWrite(leds[i], LOW);
  }
}

//function to set all stepper motor variables, outputs and LEDs
void init_stepper() {
  pinMode(rtStepPin, OUTPUT);                 //sets pin as output
  pinMode(rtDirPin, OUTPUT);                  //sets pin as output
  pinMode(ltStepPin, OUTPUT);                 //sets pin as output
  pinMode(ltDirPin, OUTPUT);                  //sets pin as output
  pinMode(stepperEnable, OUTPUT);             //sets pin as output
  digitalWrite(stepperEnable, stepperEnFalse);  //turns off the stepper motor driver
  pinMode(enableLED, OUTPUT);                 //set enable LED as output
  digitalWrite(enableLED, LOW);               //turn off enable LED
  pinMode(redLED, OUTPUT);                    //set red LED as output
  pinMode(grnLED, OUTPUT);                    //set green LED as output
  pinMode(ylwLED, OUTPUT);                    //set yellow LED as output
  digitalWrite(redLED, HIGH);               //turn on red LED
  digitalWrite(ylwLED, HIGH);               //turn on yellow LED
  digitalWrite(grnLED, HIGH);               //turn on green LED
  delay(pauseTime / 5);                   //wait 0.5 seconds
  digitalWrite(redLED, LOW);                //turn off red LED
  digitalWrite(ylwLED, LOW);                //turn off yellow LED
  digitalWrite(grnLED, LOW);                //turn off green LED
  stepperRight.setMaxSpeed(max_speed);       //set the maximum permitted speed limited by processor
and clock speed, no greater than 4000 steps/sec on Arduino
  stepperRight.setAcceleration(max_accel);    //set desired acceleration in steps/s^2
  stepperLeft.setMaxSpeed(max_speed);        //set the maximum permitted speed limited by processor
and clock speed, no greater than 4000 steps/sec on Arduino
  stepperLeft.setAcceleration(max_accel);     //set desired acceleration in steps/s^2
  steppers.addStepper(stepperRight);         //add right motor to MultiStepper
  steppers.addStepper(stepperLeft);          //add left motor to MultiStepper
  digitalWrite(stepperEnable, stepperEnTrue);  //turns on the stepper motor driver
  digitalWrite(enableLED, HIGH);             //turn on enable LED
}

//function prints encoder data to serial monitor
void print_encoder_data() {
  static unsigned long timer = 0;                          //print manager timer
  if (millis() - timer > 100) {                          //print encoder data every 100 ms or so
    lastSpeed[LEFT] = encoder[LEFT];                      //record the latest left speed value
    lastSpeed[RIGHT] = encoder[RIGHT];                    //record the latest right speed value
    accumTicks[LEFT] = accumTicks[LEFT] + encoder[LEFT];     //record accumulated left ticks
    accumTicks[RIGHT] = accumTicks[RIGHT] + encoder[RIGHT];  //record accumulated right ticks
    Serial.println("Encoder value:");
    Serial.print("\tLeft:\t");
    Serial.print(encoder[LEFT]);
    Serial.print("\tRight:\t");
    Serial.println(encoder[RIGHT]);
    Serial.println("Accumulated Ticks: ");
    Serial.print("\tLeft:\t");
    Serial.print(accumTicks[LEFT]);
    Serial.print("\tRight:\t");
    Serial.println(accumTicks[RIGHT]);
    encoder[LEFT] = 0;   //clear the left encoder data buffer
    encoder[RIGHT] = 0;  //clear the right encoder data buffer
```

```
   timer = millis();    //record current time since program started
  }
}

/*this function will reset the encoders*/
void resetEncoder() {
  encoder[LEFT] = 0;   //clear the left encoder data buffer
  encoder[RIGHT] = 0;  //clear the right encoder data buffer
}

/*this function will stop the steppers*/
void stopMove() {
  stepperRight.stop();  // Stop right motor
  stepperLeft.stop();   // Stop left motor
}

/*
  This function performs PID control to adjust the position of the robot's wheels.
  Inputs:
    - leftDistance: Target distance for the left wheel.
    - rightDistance: Target distance for the right wheel.
  The function calculates the error between the target and actual encoder values,
  applies a proportional control to determine the necessary movement adjustments,
  and sets the wheel speeds accordingly. The steppers are then run to the computed positions.
  If the error is within a specified threshold, the function stops the steppers.
*/
void PIDControl(int leftDistance, int rightDistance) {
  // Calculate the error between the target and actual encoder values
  long leftDistanceError = abs(leftDistance) - encoder[LEFT] * encoderRatio;
  long rightDistanceError = abs(rightDistance) - encoder[RIGHT] * encoderRatio;

  // Apply proportional control
  if (abs(leftDistanceError) > PIDThreshold || abs(rightDistanceError) > PIDThreshold) {
    long outputLeft = leftDistanceError * PIDkp;
    long outputRight = rightDistanceError * PIDkp;

    // Set the wheel speeds
    if (leftDistanceError < 0) {
      stepperLeft.setSpeed(-stepperLeft.speed());
    }
    if (rightDistanceError < 0) {
      stepperRight.setSpeed(-stepperRight.speed());
    }

    // Run the steppers
    stepperLeft.move(outputLeft);
    stepperRight.move(outputRight);
    steppers.runSpeedToPosition();
  }
  stopMove();
}

/**
 * Converts a length in inches to a number of steps for the robot.
 * @param length the length in inches to convert
 * @return the number of steps for the robot to move the given length
 */

int length2Steps(double length) {
  return round(stepsPerRevol * length / (2 * pi * wheelRadius));
}

/**
 * Spins the robot about its center by rotating both wheels in opposite directions.
 * @param direction the direction of the spin (TO_LEFT or TO_RIGHT)
 * @param angle the angle of the spin in degrees
 * @param speed the speed of the spin in steps per second
 */

void spin(int direction, double angle, int speed) {
  if (direction != TO_LEFT && direction != TO_RIGHT && angle < 0) return;
  resetEncoder();

  // calculate the turn length, using the circle formula
  double stepperMoveLength = pi * robotWidth * angle / 360;
  int stepperMovePos = length2Steps(stepperMoveLength);

  if (direction == TO_RIGHT) {
    stepperLeft.move(stepperMovePos);
    stepperLeft.setSpeed(speed);

    stepperRight.move(-stepperMovePos);
    stepperRight.setSpeed(-speed);
  } else if (direction == TO_LEFT) {
    stepperLeft.move(-stepperMovePos);
    stepperLeft.setSpeed(-speed);

    stepperRight.move(stepperMovePos);
    stepperRight.setSpeed(speed);
  }

  steppers.runSpeedToPosition();

  // use Encoder PID Control
  if (direction == TO_RIGHT) {
    PIDControl(stepperMovePos, -stepperMovePos);
  } else if (direction == TO_LEFT) {
    PIDControl(-stepperMovePos, stepperMovePos);
  }
}

/**
 * Moves the robot forward by a specified distance at a specified speed.
 * @param distance the distance in inches to move the robot
 * @param speed the speed in steps per second to move the robot
 */
void forward(double distance, int speed) {
  resetEncoder();
  int stepperMovePos = length2Steps(distance);
  stepperLeft.setCurrentPosition(0);
  stepperRight.setCurrentPosition(0);

  stepperLeft.move(stepperMovePos);   //move left wheel to relative position
  stepperRight.move(stepperMovePos);  //move right wheel to relative position

  stepperLeft.setSpeed(speed);   //set left motor speed
  stepperRight.setSpeed(speed);  //set right motor spee
  steppers.runSpeedToPosition();

  PIDControl(stepperMovePos, stepperMovePos);
}

/**
 * Moves the robot backward by a specified distance at a specified speed.
 * @param distance the distance in inches to move the robot backward
 * @param speed the speed in steps per second to move the robot backward
 * The function calls the forward() function with negative values of distance and speed.
 */
```

```
void reverse(double distance, int speed) {
  forward(-distance, -speed);
}

/**
 * Turns the robot to the absolute angle specified.
 * @param angle the absolute angle to turn to in degrees
 * @note Positive angles are to the left, negative angles are to the right
 */
void goToAngle(double angle) {
  // Serial.println("goToAngle function");

  // digitalWrite(grnLED, HIGH);  //turn on green LED
  // if angle larger than 0, turn left
  // if angle less than 0, turn right
  int direction = 0;
  if (angle > 0) {
    direction = TO_LEFT;
  } else if (angle < 0) {
    direction = TO_RIGHT;
  } else {
    return;
  }
  angle = abs(angle);
  spin(direction, angle, defaultStepSpeed);
}

/**
 * @brief Calculates the turn angle in degrees based on the given x and y coordinates.
 *
 * This function computes the angle in radians using the arctangent of y/x, then converts it to degrees.
 * It adjusts the angle based on the quadrant of the (x, y) point to ensure the correct angle is returned.
 *
 * @param x The x-coordinate.
 * @param y The y-coordinate.
 * @return The turn angle in degrees.
 */
double getTurnAngle(double x, double y) {
  double angleRadian = atan(y / x);

  double angleDegree = angleRadian * 180.0 / pi;

  if (x > 0 && y > 0) {
    angleDegree = angleDegree;
  } else if (x < 0 && y >= 0) {
    angleDegree = 180 + angleDegree;
  } else if (x < 0 && y < 0) {
    angleDegree = 180 + angleDegree;
  } else if (x > 0 && y < 0) {
    angleDegree = angleDegree;
  }

  return angleDegree;
}

/**
 * Moves the robot to the goal location (x, y) in inches.
 * @param x the x-coordinate of the goal in inches
 * @param y the y-coordinate of the goal in inches
 * @note The robot will first turn to the correct angle, then move forward to the goal
 */
void goToGoal(double x, double y) {
  // Serial.println("goToGoal function");
  digitalWrite(redLED, LOW);   //turn off red LED
  digitalWrite(grnLED, HIGH);  //turn on green LED
  digitalWrite(ylwLED, HIGH);  //turn on yellow LED

  double distance = sqrt(pow(x, 2) + pow(y, 2));
  double angleDegree = getTurnAngle(x, y);

  Serial.print("Angle: ");
  Serial.println(angleDegree);

  goToAngle(angleDegree);
  delay(1000);
  forward(distance, defaultStepSpeed);
}

/**
 * Prints the random values generated for the angle and distance.
 * @param randAngle the random angle generated
 * @param randDistance the random distance generated
 */
void printRandomValues(int randAngle, int randDistance) {
  Serial.print("Random Values:\n\tAngle: ");
  Serial.print(randAngle);
  Serial.print("\tDistance: ");
  Serial.println(randDistance);
}

/**
 * Generates a random angle and distance for the robot to move.
 * The robot will turn to the random angle, then move forward the random distance.
 */
void randomWander() {
  digitalWrite(grnLED, HIGH);  //turn on green LED

  int randAngle = random(-maxTurnAngle, maxTurnAngle);
  int randDistance = random(maxDistanceMove);

  goToAngle(randAngle);
  forward(randDistance, defaultStepSpeed);

  // printRandomValues(randAngle, randDistance);
}


// reads a lidar given a pin
int read_lidar(int pin) {
  // Wait for pulse
  int16_t t = pulseIn(pin, HIGH);

  int d = MAX_LIDAR_DISTANCE;  // Default to "no object"
  if (t != 0 && t <= 1850) {
    d = (t - 1000) * 3 / 40;
    if (d < 0) d = 0;
  }

  delay(10);  // More stable than delayMicroseconds
  return d;
}

// reads a sonar given a pin
int read_sonar(int pin) {
  float velocity((331.5 + 0.6 * (float)(20)) * 100 / 1000000.0);
  uint16_t distance, pulseWidthUs;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  digitalWrite(pin, HIGH);          //Set the trig pin High
  delayMicroseconds(10);             //Delay of 10 microseconds
  digitalWrite(pin, LOW);           //Set the trig pin Low
  pinMode(pin, INPUT);              //Set the pin to input mode
  pulseWidthUs = pulseIn(pin, HIGH);  //Detect the high level time on the echo pin, the output high level
time represents the ultrasonic flight time (unit: us)
  distance = pulseWidthUs * velocity / 2.0;
  if (distance < 0 || distance > 50) { distance = 0; }
  return distance;
}

// prints the sensor data
void print_sensor_data(struct lidar data, struct sonar data2) {
  Serial.print("lidar: ");
  Serial.print(data.front);
  Serial.print(", ");
  Serial.print(data.back);
  Serial.print(", ");
  Serial.print(data.left);
  Serial.print(", ");
  Serial.print(data.right);
  Serial.println();
  // Serial.print("sonar: ");
  // Serial.print(data2.left);
  // Serial.print(", ");
  // Serial.print(data2.right);
  // Serial.println();
}

// converts cm to inches
```

```
double cm2inch(int cm) {
  return 0.393701 * cm;
}

// collide behavior
void collideBehavior() {
  digitalWrite(redLED, HIGH);
  digitalWrite(ylwLED, LOW);
  digitalWrite(grnLED, LOW);
  stopMove();
  delay(50);  // Small delay to prevent CPU hogging
}

// check if there is an obstacle around the robot
bool isCloseObstacle() {
  // Read sensor data
  lidar_data = RPC.call("read_lidars").as<struct lidar>();
  sonar_data = RPC.call("read_sonars").as<struct sonar>();

  // Return true if any sensor detects a close obstacle
  return frontHasObstacle() || backHasObstacle() || leftHasObstacle() || rightHasObstacle();
  // return (sonar_data.left < OBSTACLE_THRESHOLD && sonar_data.left != 0) ||
  //      (sonar_data.right < OBSTACLE_THRESHOLD && sonar_data.right != 0);
}

// check if there is an obstacle in front of the robot
bool checkFrontObstacle() {
  lidar_data = RPC.call("read_lidars").as<struct lidar>();
  return frontHasObstacle();
}

// check if there is an obstacle in the front of the robot
```

```
bool frontHasObstacle() {
  return lidar_data.front < OBSTACLE_THRESHOLD;
}

// check if there is an obstacle in the back of the robot
bool backHasObstacle() {
  return lidar_data.back < OBSTACLE_THRESHOLD;
}

// check if there is an obstacle on the left of the robot
bool leftHasObstacle() {
  return lidar_data.left < OBSTACLE_THRESHOLD;
}

// check if there is an obstacle on the right of the robot
bool rightHasObstacle() {
  return lidar_data.right < OBSTACLE_THRESHOLD;
}

// Runaway behavior
const double runawayPropotion = 0.5;
const int forwardDistance = 10;

/**
 * @brief Executes the runaway behavior for the robot.
 *
 * This function controls the robot's movement based on sensor data to avoid obstacles.
 * It reads lidar data, calculates the necessary turn angle, and moves the robot accordingly.
 *
 * The function performs the following steps:
 * 1. Turns on the yellow LED and turns off the red and green LEDs.
 * 2. Reads lidar data using an RPC call.
 * 3. Calculates the x and y distances in inches based on the lidar data.
 * 4. Determines the turn angle based on the presence of obstacles detected by the sensors.
 * 5. Moves the robot to the calculated angle and moves forward if the turn angle is valid.
 * 6. Stops the robot if the turn angle is invalid.
 *
 * The function uses the following helper functions:
 * - frontHasObstacle(): Checks if there is an obstacle in front of the robot.
 * - backHasObstacle(): Checks if there is an obstacle behind the robot.
 * - leftHasObstacle(): Checks if there is an obstacle to the left of the robot.
 * - rightHasObstacle(): Checks if there is an obstacle to the right of the robot.
 * - getTurnAngle(double x_inch, double y_inch): Calculates the turn angle based on x and y distances.
 * - goToAngle(double angle): Turns the robot to the specified angle.
 * - forward(double distance, int speed): Moves the robot forward by the specified distance at the given
speed.
 * - stopMove(): Stops the robot's movement.
 */
void runawayBehavior() {
  digitalWrite(ylwLED, HIGH);
  digitalWrite(redLED, LOW);
  digitalWrite(grnLED, LOW);
  lidar_data = RPC.call("read_lidars").as<struct lidar>();

  int x = lidar_data.front - lidar_data.back;
  int y = lidar_data.left - lidar_data.right;

  double x_inch = runawayPropotion * cm2inch(x);
  double y_inch = runawayPropotion * cm2inch(y);

  // print_sensor_data(lidar_data, sonar_data);

  double turnAngle = 0;
  if (!frontHasObstacle() && !backHasObstacle() && leftHasObstacle() && rightHasObstacle()) {
    turnAngle = 0;
  } else if (!leftHasObstacle() && !rightHasObstacle() && frontHasObstacle() && backHasObstacle()) {
    turnAngle = 90.0;
  } else if (frontHasObstacle() && backHasObstacle() && leftHasObstacle() && rightHasObstacle()) {
    turnAngle = 3600;
  } else if (!frontHasObstacle() && !backHasObstacle() && !leftHasObstacle() && !rightHasObstacle()){
    return;
  } else {
    turnAngle = getTurnAngle(x_inch, y_inch);
  }

  if(turnAngle <= 360){
    goToAngle(turnAngle);
    forward(forwardDistance, defaultStepSpeed);
  }else{
    stopMove();
  }
}

const double followPropotion = 0.25;
const double OBSTACLE_MARGIN = 2.0;
const double FOLLOW_SPEED = 200;

// Check if the robot is within the obstacle margin
bool isWithinObstacleMargin(double frontDistance) {
  return frontDistance < OBSTACLE_THRESHOLD + OBSTACLE_MARGIN && frontDistance >
OBSTACLE_THRESHOLD - OBSTACLE_MARGIN;
}

/**
 * @brief Executes the follow behavior for a robot using LIDAR data.
 *
 * This function controls the robot to follow an object by maintaining a certain distance
 * from it. It uses LIDAR data to determine the distance to the object and adjusts the
 * robot's movement accordingly. The function continuously reads LIDAR data and moves
 * the robot forward or backward to maintain the desired distance.
 *
 * The function performs the following steps:
 * 1. Turns on the red and green LEDs and turns off the yellow LED.
 * 2. Reads the initial LIDAR data and calculates the front error in centimeters and inches.
 * 3. Enters an infinite loop where it:
 *    - Checks if the object is within the obstacle margin.
 *    - If the object is not within the obstacle margin and is within the maximum LIDAR distance:
 *      - Calculates the movement distance based on the proportional control.
 *      - Moves the robot forward if the front error is positive, or backward if the front error is negative.
 *    - Reads the updated LIDAR data and recalculates the front error.
 *    - Delays for 20 milliseconds before the next iteration.
 *
 * The function exits the loop and stops the behavior if the object is beyond the maximum LIDAR
distance.
 */
void followBehavior() {
  digitalWrite(redLED, HIGH);
  digitalWrite(grnLED, HIGH);
  digitalWrite(ylwLED, LOW);

  lidar_data = RPC.call("read_lidars").as<struct lidar>();
  int front_error_cm = lidar_data.front - OBSTACLE_THRESHOLD;
  double front_error_inch = cm2inch(front_error_cm);

  while (true) {
    // Serial.print("FRONT ERROR: ");
    // Serial.println(front_error_cm);
```

```
   // Serial.print("FRONT MOVE: ");
   // Serial.println(followPropotion * front_error_inch);

   if (!isWithinObstacleMargin(lidar_data.front)) {
    if (lidar_data.front >= MAX_LIDAR_DISTANCE) break;

    double moveDistance = abs(followPropotion * front_error_inch);

    if (front_error_cm > 0) {
     forward(moveDistance, FOLLOW_SPEED);
    } else if (front_error_cm < 0) {
     reverse(moveDistance, FOLLOW_SPEED);
    }
   }

   lidar_data = RPC.call("read_lidars").as<struct lidar>();
   front_error_cm = lidar_data.front - OBSTACLE_THRESHOLD;
   front_error_inch = cm2inch(front_error_cm);
   delay(20);
  }
}

/**
 * @brief Executes the smart wander behavior for a robot.
 *
 * This function makes the robot move in a random direction and distance,
 * while continuously checking for obstacles. If an obstacle is detected,
 * the robot will stop, perform a collision avoidance behavior, and then
 * resume wandering.
 *
 * The function performs the following steps:
```

```
 * 1. Generates a random angle and distance for the robot to move.
 * 2. Converts the distance to steps for the stepper motors.
 * 3. Rotates the robot to the random angle.
 * 4. Moves the robot forward by the random distance.
 * 5. Sets the speed of the stepper motors to the default speed.
 * 6. Continuously runs the stepper motors and checks for obstacles.
 * 7. If an obstacle is detected, stops the motors, performs collision
 *    avoidance, and resumes wandering.
 * 8. Turns on the green LED while moving, and turns off other LEDs.
 * 9. Adds a small delay to allow sensor readings to be processed.
 */
void smartWanderBehavior() {

  int randAngle = random(-maxTurnAngle, maxTurnAngle);
  int randDistance = random(maxDistanceMove);
  randDistance = length2Steps(randDistance);

  goToAngle(randAngle);

  stepperLeft.move(randDistance);
  stepperRight.move(randDistance);

  stepperLeft.setSpeed(defaultStepSpeed);
  stepperRight.setSpeed(defaultStepSpeed);

  while (steppers.run()) {
    digitalWrite(grnLED, HIGH);  //turn on green LED
    digitalWrite(ylwLED, LOW);
    digitalWrite(redLED, LOW);

    while (isCloseObstacle()) {
      // Obstacle detected - stop motors and turn on LED
      collideBehavior();
      delay(100);
      runawayBehavior();
    }
  }

  // Small delay to allow sensor readings to be processed
  delay(1);
}

/**
 * @brief Executes the smart follow behavior for a robot.
 *
 * This function makes the robot move in a random direction and distance,
 * while continuously checking for obstacles. If an obstacle is detected,
 * the robot will stop and execute the collide behavior, followed by the
 * follow behavior.
 *
 * The function performs the following steps:
 * 1. Generates a random angle and distance for the robot to move.
 * 2. Converts the distance to steps for the stepper motors.
 * 3. Rotates the robot to the generated angle.
 * 4. Moves the robot forward by the generated distance.
 * 5. Sets the speed of the stepper motors to the default speed.
 * 6. Continuously runs the stepper motors while checking for obstacles.
 * 7. If an obstacle is detected, stops the motors and executes the collide behavior.
 * 8. Turns on the green LED to indicate movement.
 * 9. Adds a small delay to allow sensor readings to be processed.
 */
void smartFollowBehavior() {
  int randAngle = random(-maxTurnAngle, maxTurnAngle);
  int randDistance = random(maxDistanceMove);
  randDistance = length2Steps(randDistance);

  goToAngle(randAngle);

  stepperLeft.move(randDistance);
  stepperRight.move(randDistance);

  stepperLeft.setSpeed(defaultStepSpeed);
  stepperRight.setSpeed(defaultStepSpeed);

  while (steppers.run()) {
    digitalWrite(grnLED, HIGH);  //turn on green LED
    digitalWrite(ylwLED, LOW);
    digitalWrite(redLED, LOW);

    while (checkFrontObstacle()) {
      // Obstacle detected - stop motors and turn on LED
      collideBehavior();
      delay(100);
      followBehavior();
    }
  }

  // Small delay to allow sensor readings to be processed
  delay(1);
}

/**
 * @brief Initializes the stepper motor and sets up interrupts for the left and right wheel encoders.
 *
 * This function performs the following tasks:
 * - Initializes the stepper motor by calling init_stepper().
 * - Attaches an interrupt to the left wheel encoder pin, triggering the LwheelSpeed function on any
change.
 * - Attaches an interrupt to the right wheel encoder pin, triggering the RwheelSpeed function on any
change.
 * - Delays execution for 1000 milliseconds to allow for setup stabilization.
 */
void setupM7() {
  init_stepper();                                        //set up stepper motor
  attachInterrupt(digitalPinToInterrupt(ltEncoder), LwheelSpeed, CHANGE);  //init the interrupt mode for
the left encoder
  attachInterrupt(digitalPinToInterrupt(rtEncoder), RwheelSpeed, CHANGE);  //init the interrupt mode
for the right encoder
  delay(1000);
}

/**
 * @brief Controls the movement and obstacle-following behavior of the robot.
 *
 * This function initializes the LEDs and stepper motor speeds, then enters an infinite loop
 * where it continuously checks for close obstacles. If an obstacle is detected, it calls
 * the collideBehavior() and followBehavior() functions to handle the obstacle. Once the
 * obstacle is removed, it resets the LEDs and stepper motor speeds and continues moving
 * the steppers one step at a time with a small delay to allow sensor readings to be processed.
 */
void moveAndFollowBehavior() {
  digitalWrite(redLED, LOW);  // Initially LED is off
  digitalWrite(ylwLED, LOW);
  digitalWrite(grnLED, LOW);

  // Set initial movement speeds
  stepperLeft.setSpeed(defaultStepSpeed);
  stepperRight.setSpeed(defaultStepSpeed);
 while (true) {
  while (isCloseObstacle()) {
   collideBehavior();
   followBehavior();
  }
  // Obstacle removed - reset movement
  digitalWrite(redLED, LOW);
  digitalWrite(ylwLED, LOW);
  digitalWrite(grnLED, LOW);
  stepperLeft.setSpeed(defaultStepSpeed);
  stepperRight.setSpeed(defaultStepSpeed);

  // Move the steppers one step at a time
  stepperLeft.runSpeed();
  stepperRight.runSpeed();

  // Small delay to allow sensor readings to be processed
  delay(1);
 }
}

/**
 * @brief Main loop function for behavior control.
 *
 * This function initializes the state of three LEDs (red, yellow, green) to off.
 * It then enters an infinite loop where it continuously executes the
 * smartFollowBehavior function. Other behaviors such as smartWanderBehavior
 * and runawayBehavior are commented out and can be enabled if needed.
 */
void loopM7() {
 digitalWrite(redLED, LOW);  // Initially LED is off
```

```
  digitalWrite(ylwLED, LOW);
  digitalWrite(grnLED, LOW);

  while (true) {
    // smartWanderBehavior();
    smartFollowBehavior();
    // runawayBehavior();
  }
}

//set up the M4 to be the server for the sensors data
void setupM4() {
  for (int i = 0; i < numOfSens; i++) {
    pinMode(lidar_pins[i], OUTPUT);
  }
  RPC.bind("read_lidars", read_lidars);  // bind a method to return the lidar data all at once
  RPC.bind("read_sonars", read_sonars);  // bind a method to return the lidar data all at once
}

//loop for the M4 to read the sensors
void loopM4() {
  // Add delays between readings to allow sensor to stabilize
  lidarData.front = read_lidar(ft_lidar);
  delay(50);
  lidarData.back = read_lidar(bk_lidar);
  delay(50);
  lidarData.left = read_lidar(lt_lidar);
  delay(50);
  lidarData.right = read_lidar(rt_lidar);
  delay(50);

  // sonarData.right = read_sonar(leftSnr);
  // delay(20);
  // sonarData.left = read_sonar(rightSnr);
  // delay(20);
}

// MAIN
void setup() {
  int baudrate = 9600;       //serial monitor baud rate'
  randomSeed(analogRead(0));  //generate a new random number each time called
  Serial.begin(baudrate);    //start serial monitor communication
  Serial.println("Robot starting...Put ON TEST STAND");

  RPC.begin();
  if (HAL_GetCurrentCPUID() == CM7_CPUID) {
    // if on M7 CPU, run M7 setup & loop
    setupM7();
    while (1) loopM7();
  } else {
    // if on M4 CPU, run M7 setup & loop
    setupM4();
    while (1) loopM4();
  }
}

void loop() {}
```